Programming With Global Arrays and ARMCI

Vinod Tipparaju

Pacific Northwest National Laboratory

Jarek Nieplocha

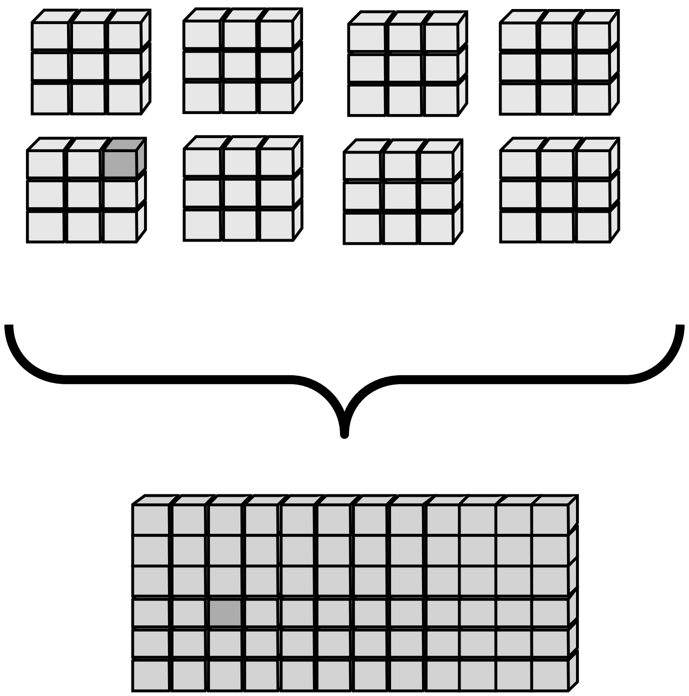
Pacific Northwest National Laboratory

Overview

- Focus on two related tools
  - Global arrays 70%
  - ARMCI 30%
- Tool overview
- Advantages and limitations
- Example
- Hands-on session

Global Arrays Overview

Physically distributed data

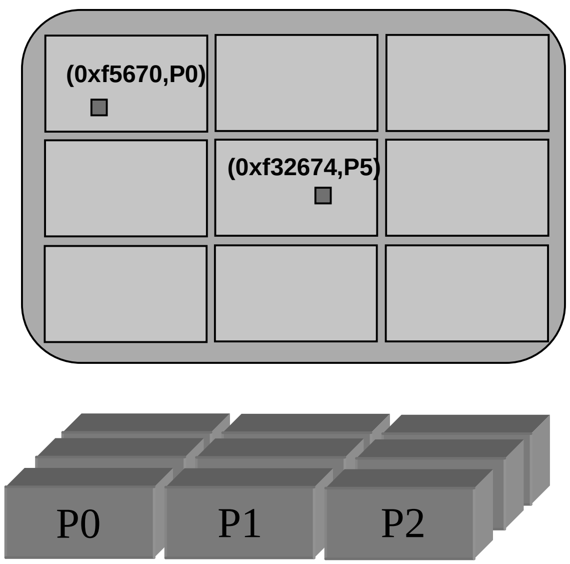


- shared memory model in context of distributed dense arrays
- complete environment for parallel code development
- compatible with MPI
- data locality control similar to distributed memory/message passing model

single, shared data structure/ global indexing
e.g., A(4,3) rather than buf(7) on task 2

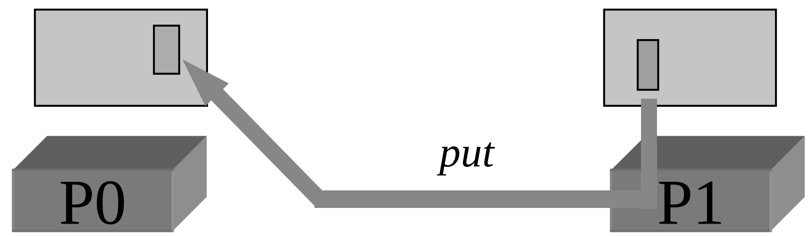
Global address space & One-sided communication

collection of address spaces
of processes in a parallel job
global address: (address, pid)



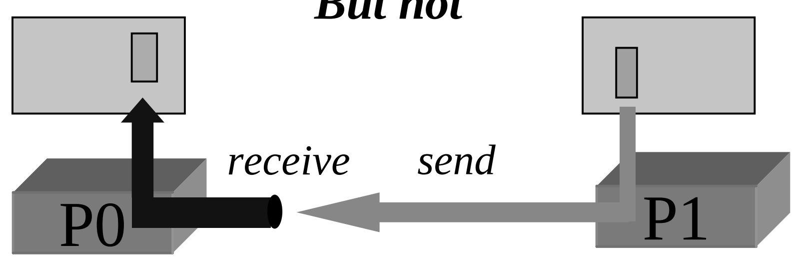
hardware examples: Cray T3E, Fujitsu VPP5000
language support: Co-Array Fortran, UPC
library support: Cray SHMEM, MPI-2, ARMCI

Communication model



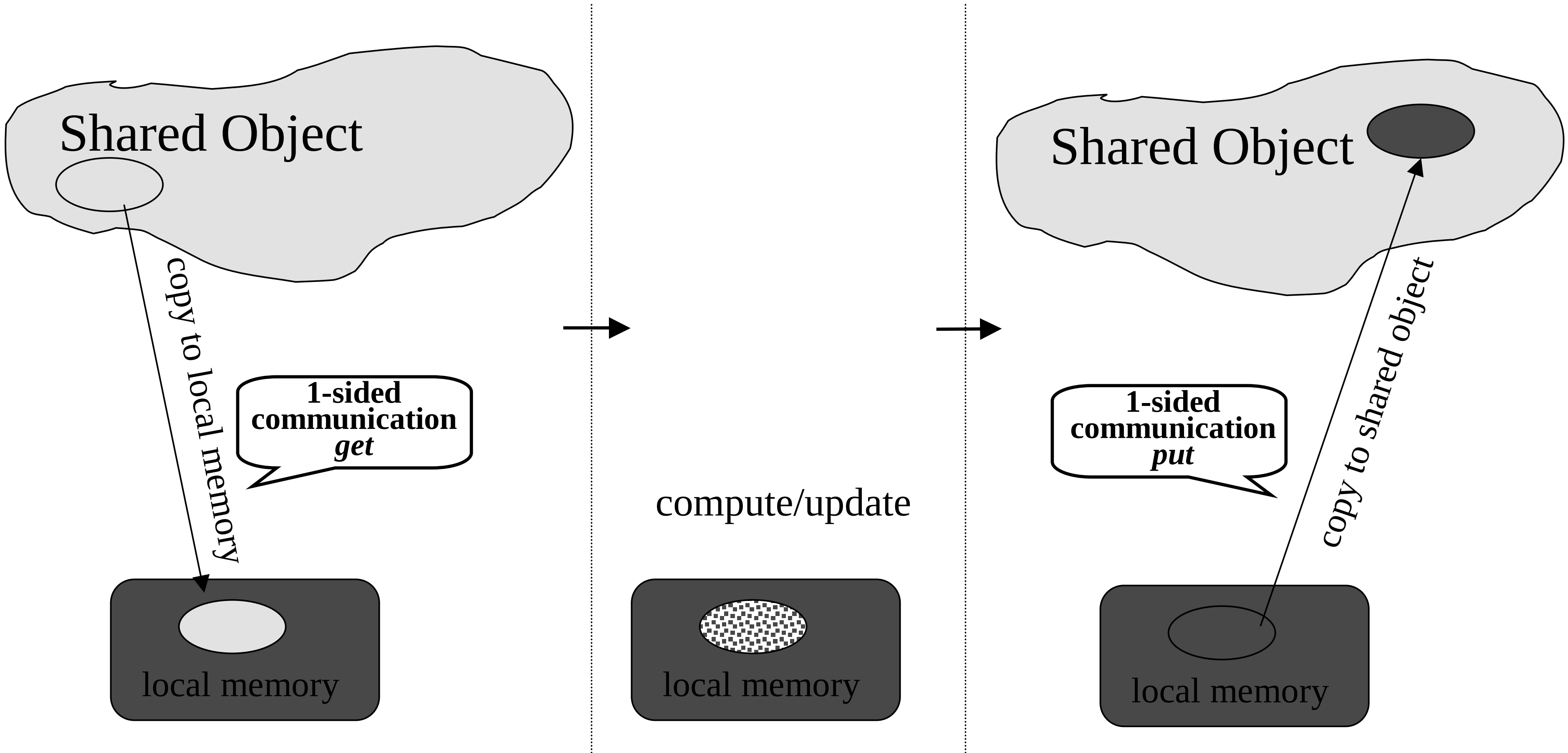
one-sided communication

But not



message passing

Global Array Model of Computations



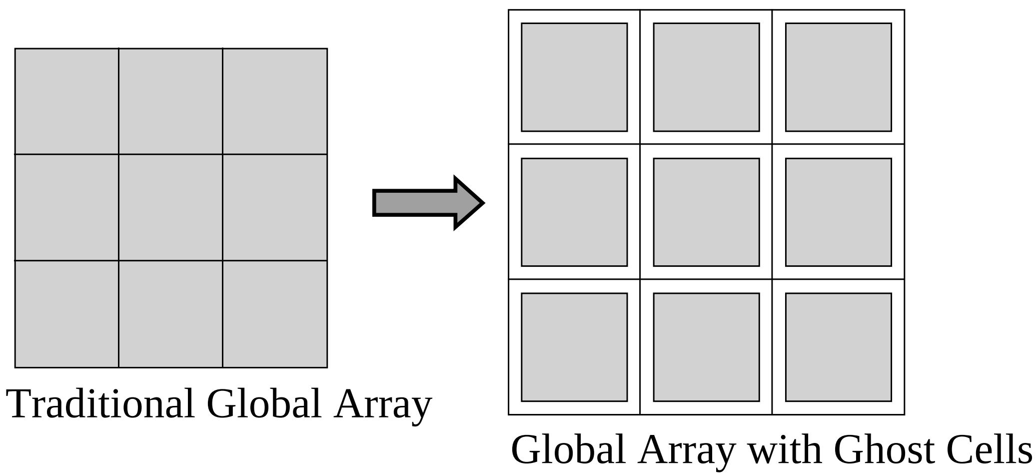
Core Capabilities

- Distributed array library
  - dense arrays 1-7 dimensions
  - three data types: *integer*, *double precision*, *double complex*
  - global rather than per-task view of data structures
  - user control over data distribution: regular and irregular
- Operations
  - collective on whole or sections of arrays
    - e.g., C(4:20,1:5) = A(1:17,2:6) + 0.5* C(4:20,2:6)
  - noncollective, 1-sided
    - *put*, *get*, *accumulate*, *scatter*, *gather*, *locks*
  - interfaces to linear algebra libraries e.g., ScaLapack, PeIGS

Language Interfaces

- Mixed language support
  - Fortran and C
  - arrays created in one language available through the others
  - native view of the data layout
- Object oriented class library interfaces
  - C++, Python
  - implemented on top of GA C interface
- Number of available operations: 115

New Capability - Ghost Cells

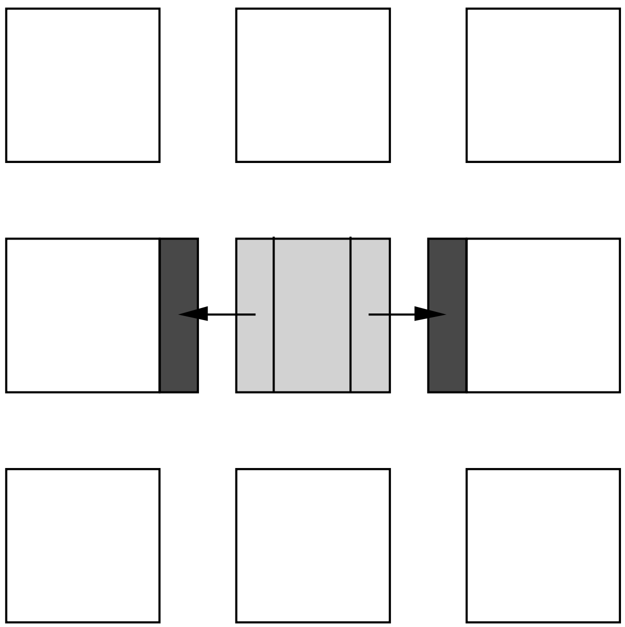


nga_create_ghosts(type, ndim, dims, width,array_name, chunk, g_a)

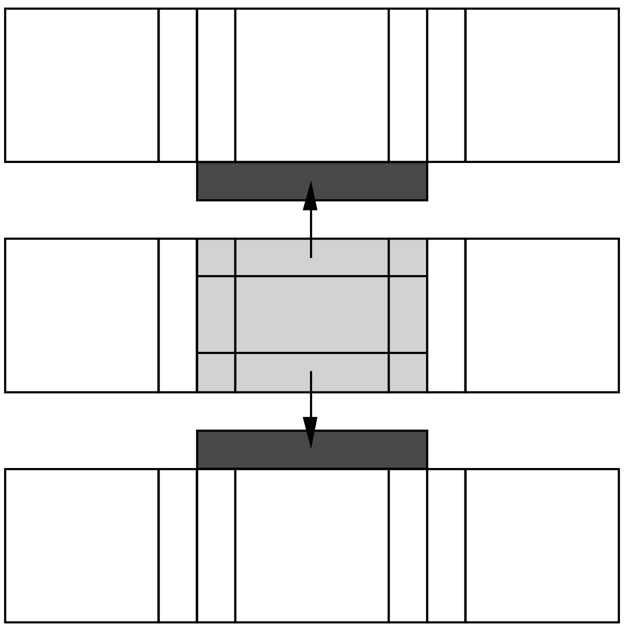
```
      character*(*) array_name: Unique character string
      integer type: Data type (MT_DBL, MT_INT, MT_DCPL)
      integer ndim: Number of array dimensions
      integer dims(ndim): Vector of array dimensions
      integer width(ndim): Vector of ghost cell widths
             integer chunk(ndim): Vector of minimum data
dimensions
                          on each processor.
      integer g_a: Integer handle for future references
```

Shift Algorithm

First sequence of updates

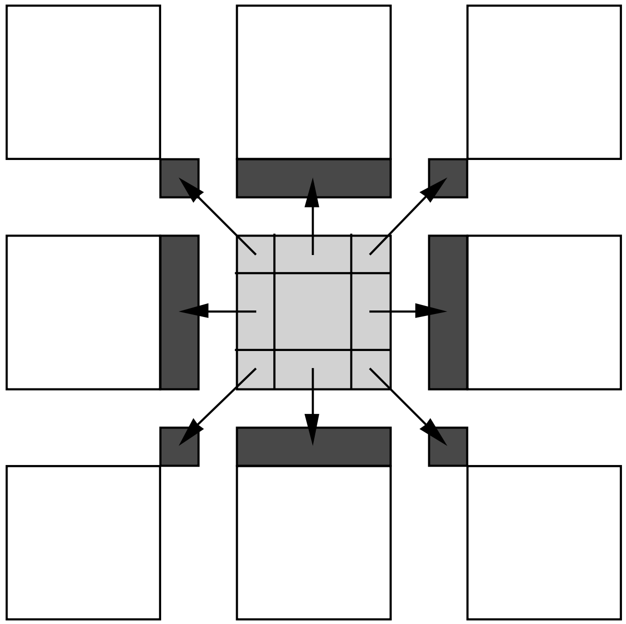


Second sequence of updates



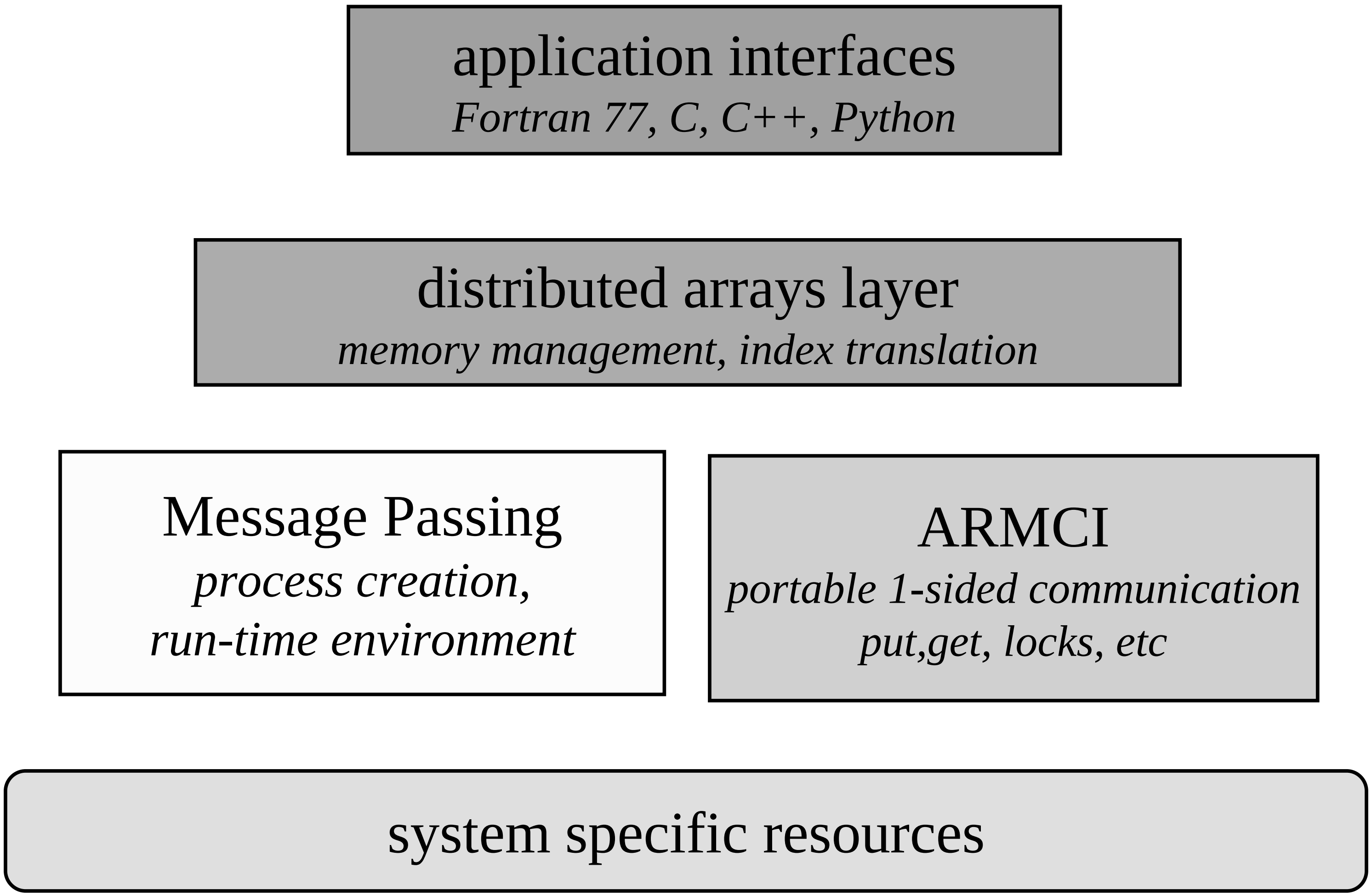
- Requires 2D messages to update ghost cells
- subroutine ga_update_ghosts(g_a)
- logical function ga_update3_ghosts(g_a)

Standard Update Algorithm

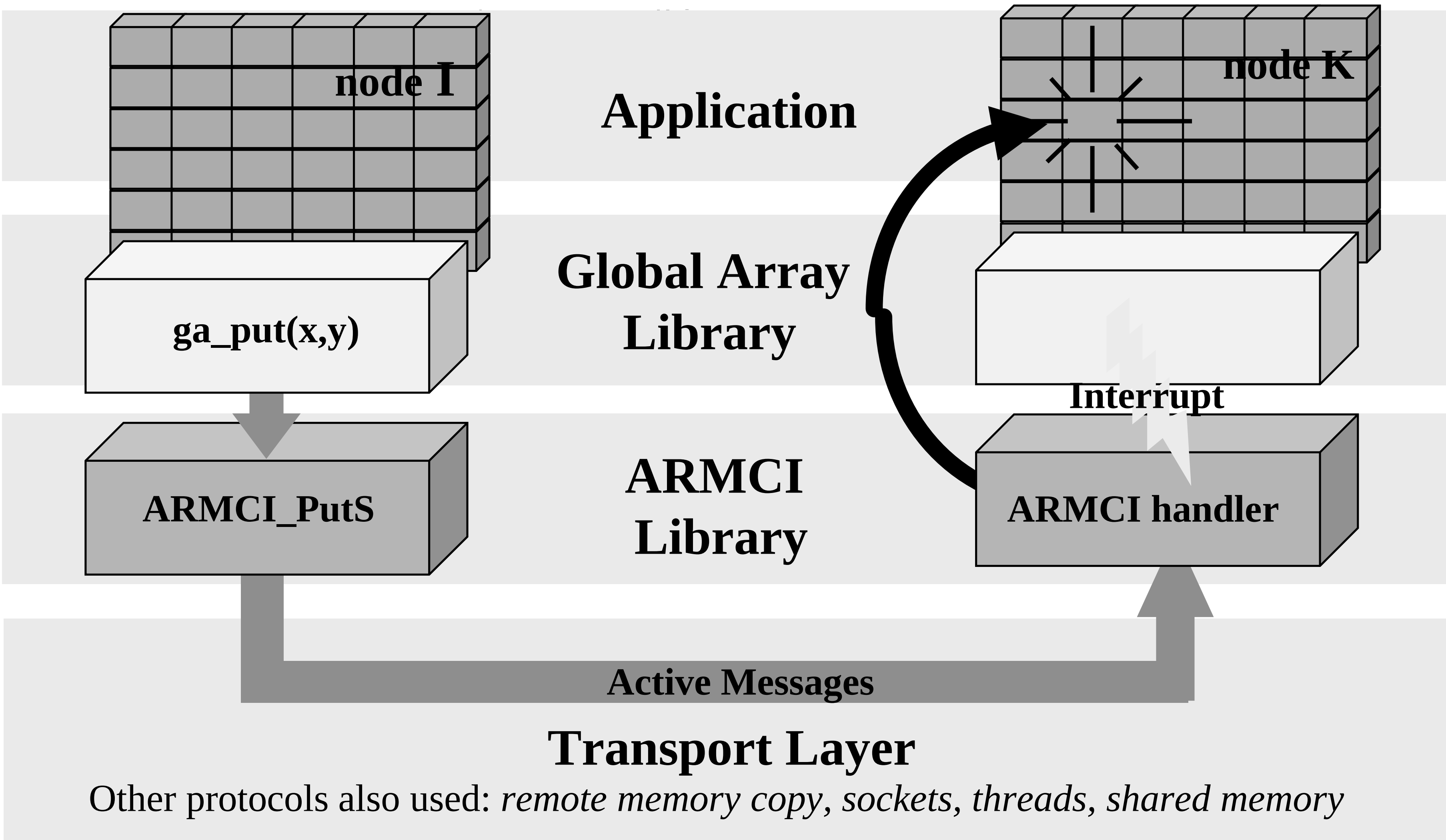


- Requires 3^D-1 messages to update ghost cells
- logical function ga_update2_ghosts(g_a)

How does GA work?

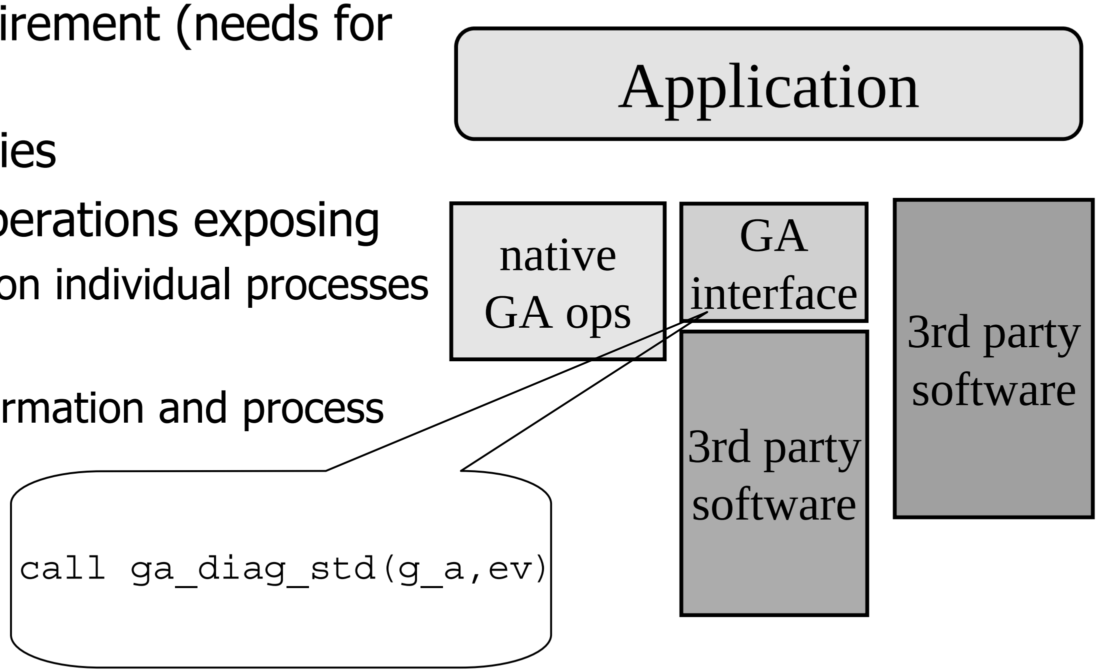


Global Array Communication via ARMCI



Interoperability

- Designed to be expandable by providing interfaces to third party software
  - application driven requirement (needs for solvers, FFT, ...)
  - message-passing libraries
  - GA provides a set of operations exposing
    - data in global arrays on individual processes
    - memory layout
    - array distribution information and process mapping



ACTS tools interoperable with GA

- linear algebra: ScaLAPACK
  - interfaces to included in GA to multiple Scalapack operations
  - example: to solve a linear system using LU factorization user calls
  -

```
call ga_lu_solve(g_a, g_b)
```

instead of

```
call pdgetrf(n,m, locA, p, q, dA, ind, info)
call pdgetrs(trans, n, mb, locA, p, q, dA, dB, info)
```

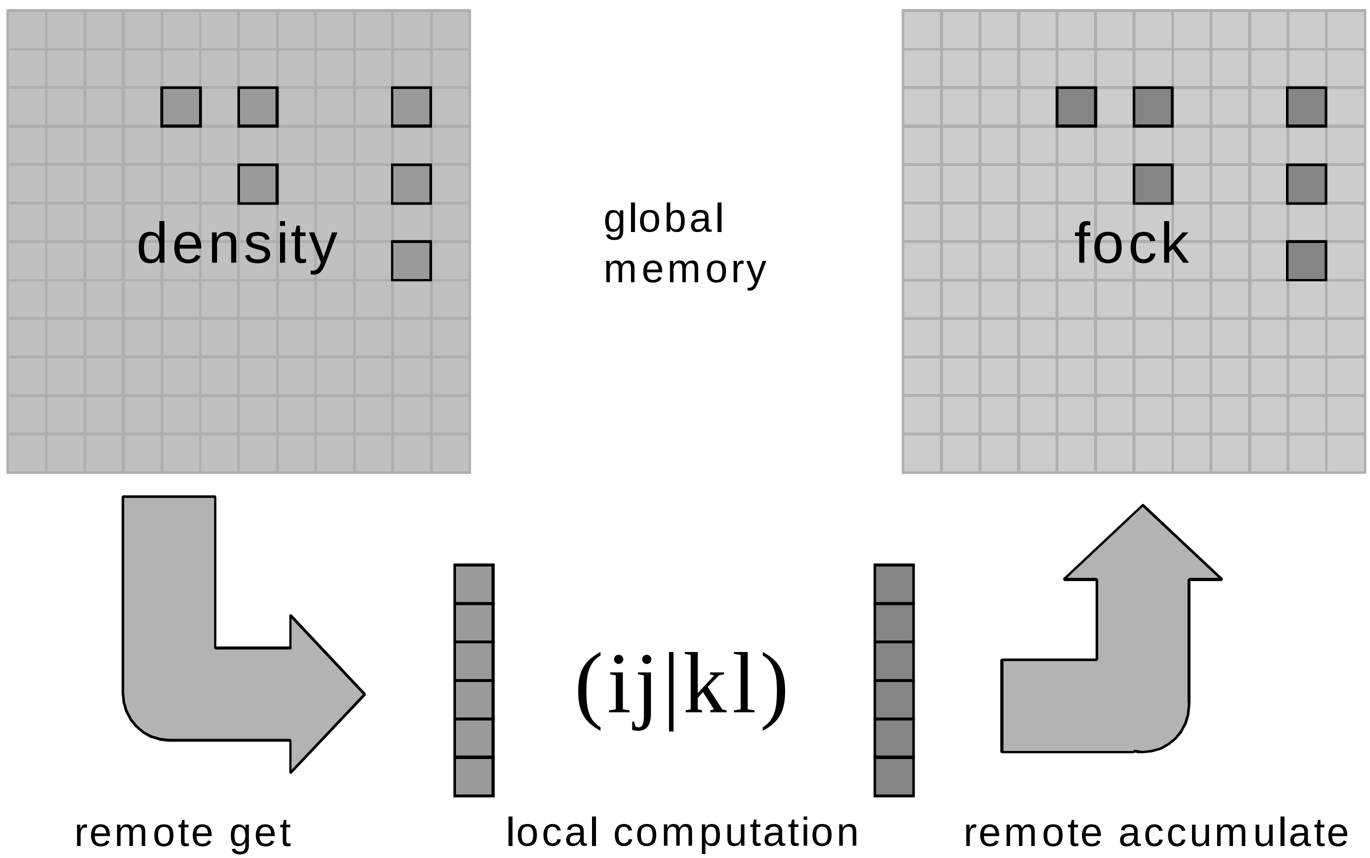
- PDE solvers: PETSc
- computational steering: CUMULUSV

Global Array Example

- Fock matrix construction
- $\mathcal{O}(N^4)$ parallelism in (ij|kl) generation - computationally intensive
- $\mathcal{O}(N^2)$ size data objects - replicate or distribute
- Case where task parallelism does not map to underlying data (cf. domain decomposition).

$$F_{kl} \mathrel{+}= (ij|kl)D_{ij}$$

Distributed-data Fock construction



GA and other models
(*biased perspective of a developer*)

	Shared memory	Message passing	MPI-2 one-sided	Global Arrays
Data view	shared	distributed	distributed	distributed or shared
Access to data	simplest (*a=b*)	Hard (*send-receive*)	moderate (*MPI_Win_Start/ MPI_Win_Post/MPI_Put/ MPI_Win_Complete*)	simple (*ga_put/get*)
Data locality information	obscured	explicit	explicit	easily available (*ga_disitribution/ ga_locate*)
Scalable performance	limited	very good	unknown (limited availability)	very good

Application guidelines

When could GA be useful?

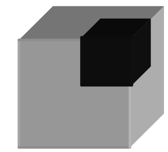
- dense distributed arrays array framework needed
- irregular communication patterns
- need one-sided access to shared data structures

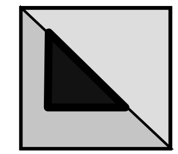
When not to use it?

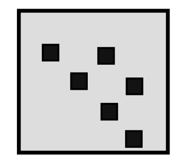
- when different data structures needed
- regular, systolic communication patterns (use MPI)
- need synchronization as a part of data transfer

ARMCI: a portable 1-sided communication library

- Functionality
  - *put*, *get, accumulate* (also with noncontiguous interfaces)
  - atomic *read-modify-write, mutexes* and *locks*
  - *memory allocation* operations
  - *fence* operations
- Characteristics
  - simple progress rules - truly one-sided
  - operations ordered w.r.t. target (ease of use)
  - less restrictive model and higher performance than MPI-2
- Applications
  - distributed array libraries: Global Arrays(PNNL), Adlib (U.Syracuse)
  - GPSHMEM - generalized portable Cray SHMEM library (Ames,PNNL)







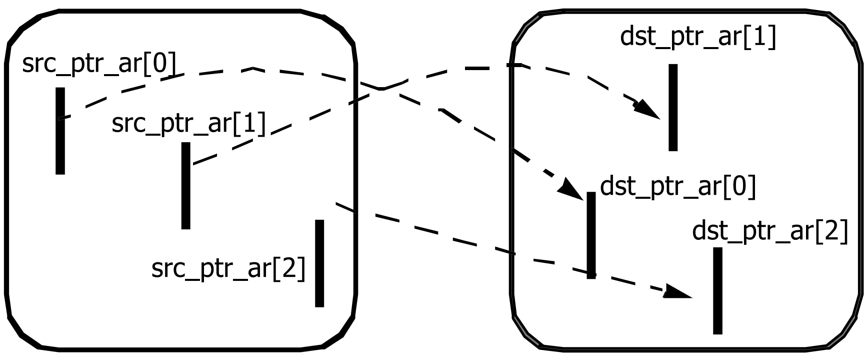
Vector API

Most general API

- based on the I/O vector API (Unix *readv*/ *writev*)
- sets of equaly-sized data segments

```
int ARMCI_PutV(armci_giov_t dscr_arr[], int arr_len,int
   proc)
    typedef struct {
          void *src_ptr_ar[];
          void *dst_ptr_ar[];
          int bytes;
          int ptr_ar_len;
    } armci_giov_t;
```

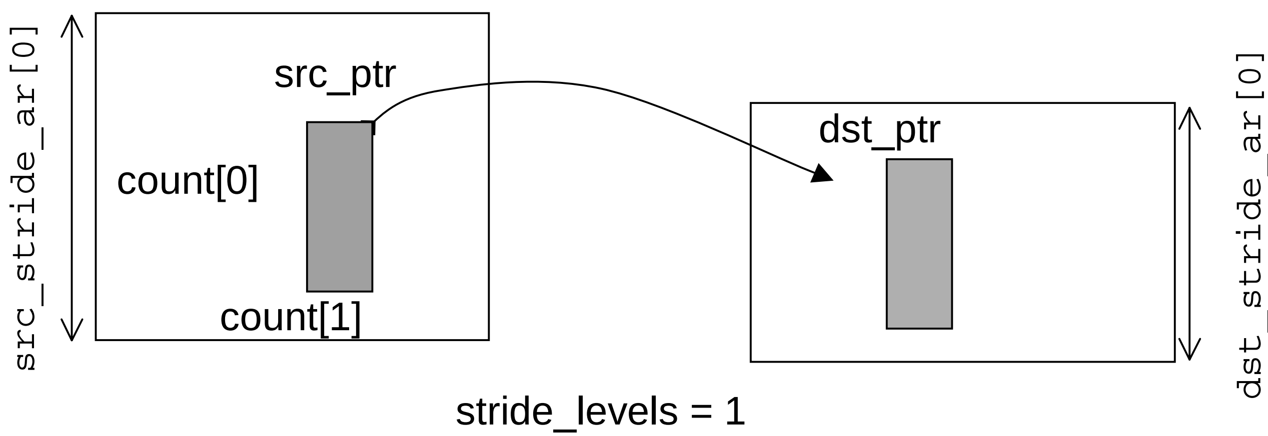
Fortran layout



Strided API

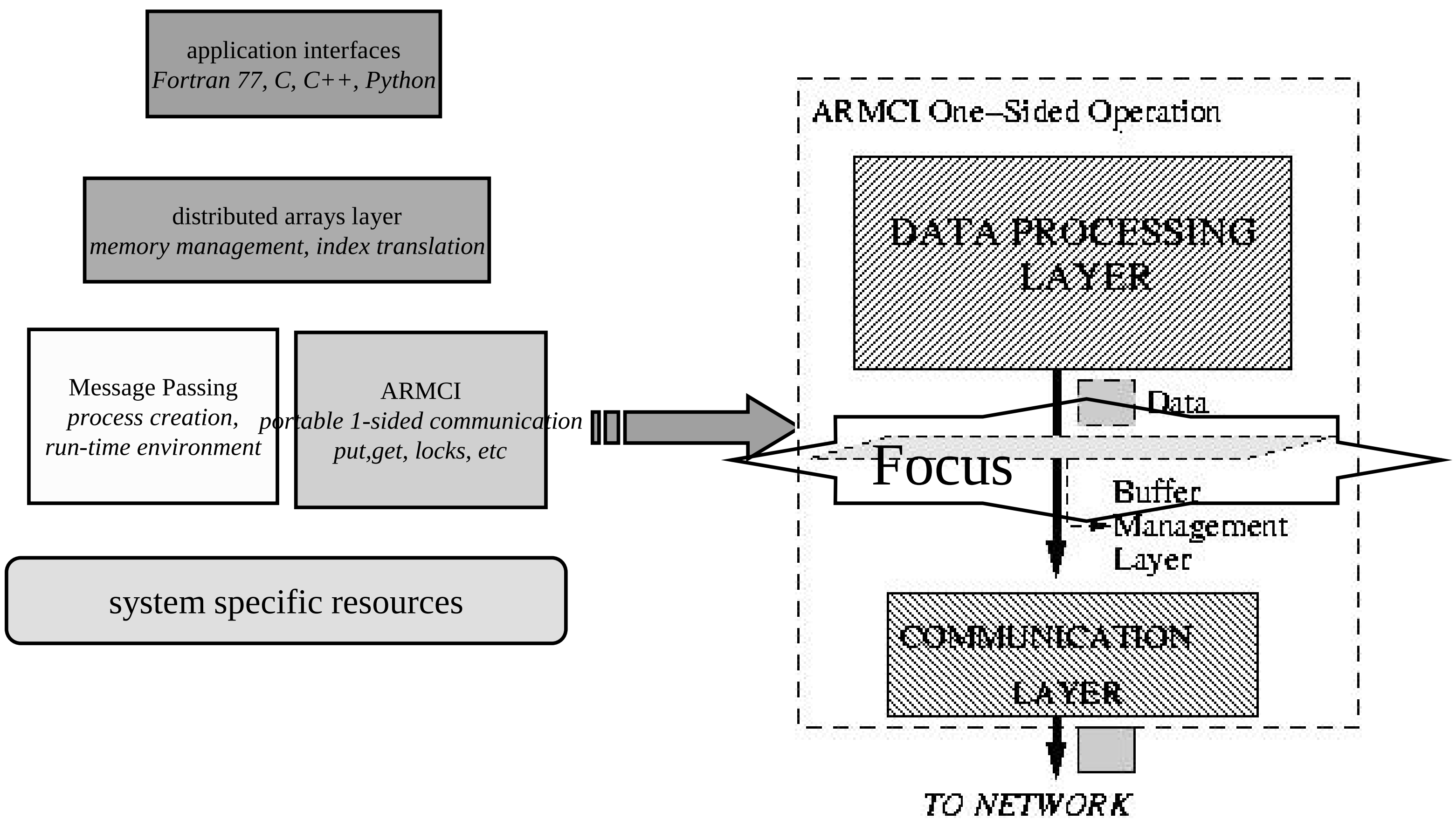
Can handle arbitrary N-dimensional array sections

```
int ARMCI_PutS(src_ptr, src_stride_ar,
               dst_ptr, dst_stride_ar, count,
               stride_levels, proc)
```



Fortran layout

OPTIMIZATIONS TO ARMCI



OPTIMIZATIONS TO ARMCI

- Optimized ARMCI_PutS operation, Additional features include
  - Efficient buffer management with pipelining
  - Adaptive sequencing of transmission buffers for optimum performance
- Optimized ARMCI_GetS operation, Additional features include
  - Dynamic determination of transmission parameters
  - Pipelined for medium to large messages
  - Current research on a model for Adaptive pipelining

Using ARMCI directly

When to use it?

- Need 1-sided communication w/o the GA infrastructure
- programmer manages distributed data structures

Advantages

- Good performance
  - e.g., 5uS latency, 320MB/s bandwidth on the NERSC Cray T3E
- Simple programming model (unlike MPI-2 1-sided)

Limitations

- Requires a message passing library to run (MPI,PVM,TCGMSG)
- Only C interfaces exist
- Memory allocation via ARMCI_Malloc

Where do I start?

- Webpage www.emsl.pnl.gov:2080/docs/global
  - User Manual user.html (*relative to the above address*)
  - C documentation Capi.html
  - Fortran documentation Fapi.html
- Download version 3.1 from the same location

- ARMCI webpage www.emsl.pnl.gov:2080/docs/parsoft/armci
  - contains links to documentation and papers
  - code distributed with Global Arrays

Features

- Separate data representation from task parallelism
- Size limited by global memory - not local memory
- Exploit full $O(N^4)$ parallelism
- Adaptable using dynamic task allocation
- Straightforward implementation